EE334

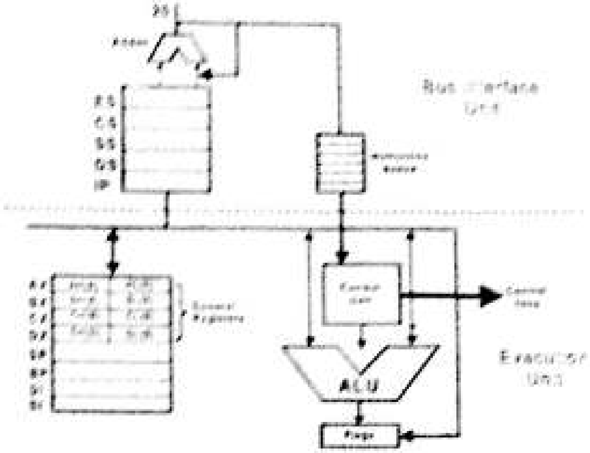
2

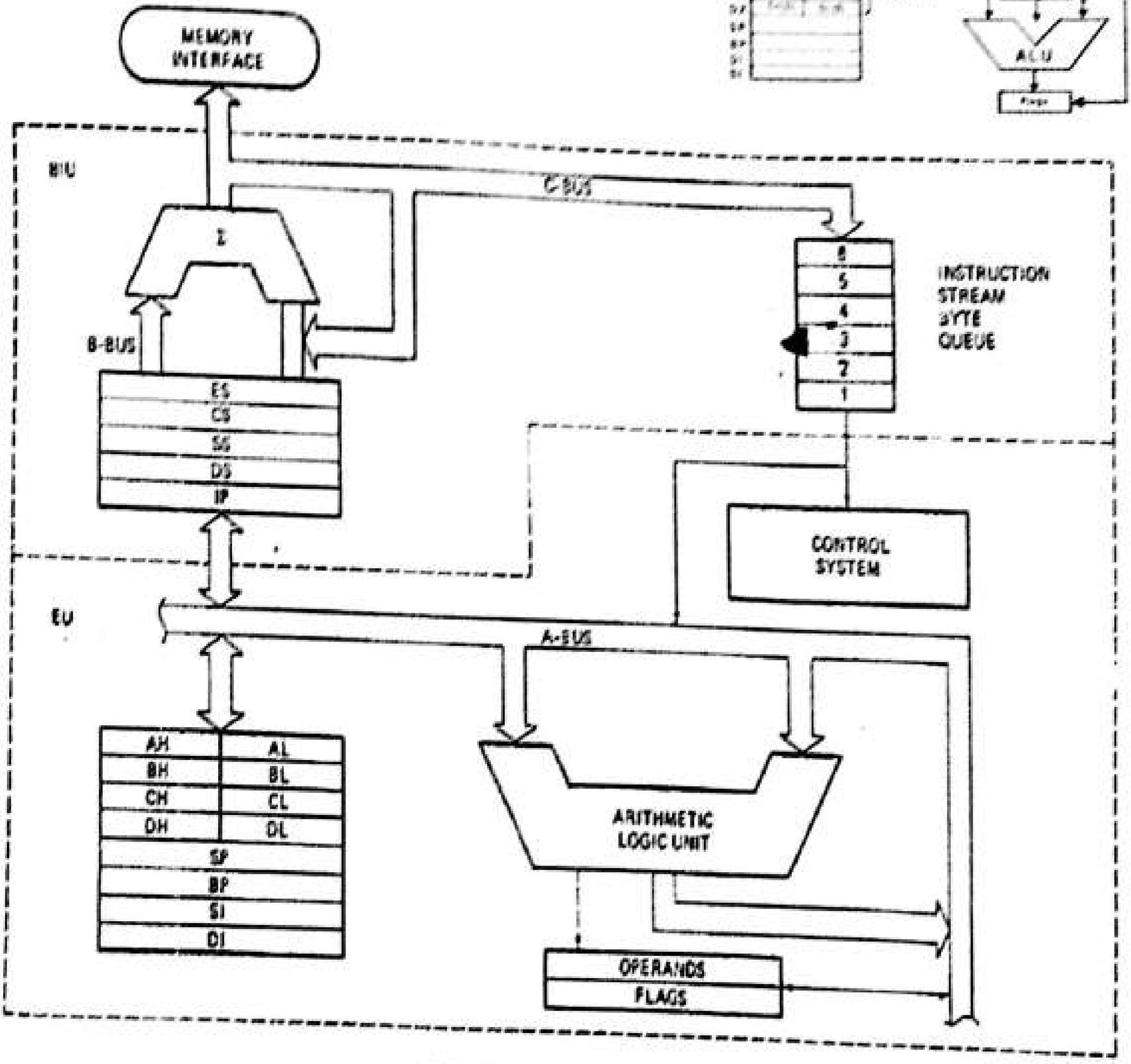
2015-3-19

Block Diagram of Intel 8086

The 8086 CPU is divided into two <u>independent</u> functional units:

1. **Bus Interface Unit (BIU)**
2. **Execution Unit (EU)**





Block Diagram of Intel 8086

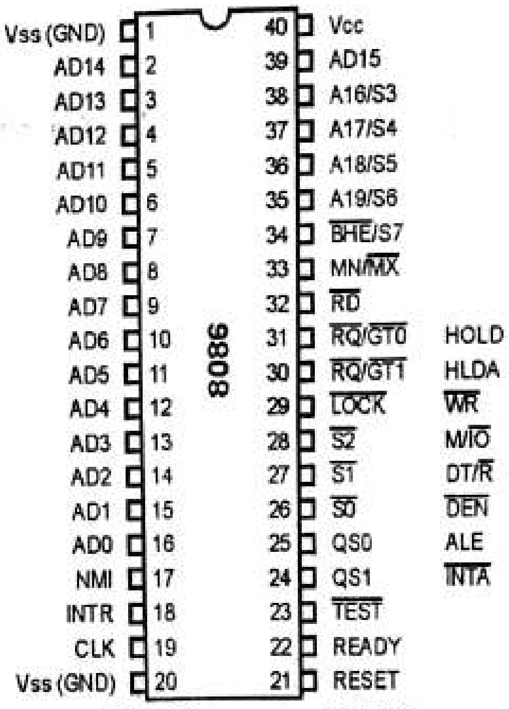
0976152324

W/1933

1

Features of 8086 Microprocessor:

1. Intel 8086 was launched in 1978.
2. It was the first 16-bit microprocessor.
3. This microprocessor had major improvement over the execution speed of 8085.
4. It requires +5V power supply.
5. It is available as 40-pin Dual-Inline-Package (DIP).
6. It consists of 29,000 transistors.
7. 8086 is a 16 bit processor. It's ALU, internal registers works with 16 bit binary word
8. 8086 has a 16 bit data bus. It can read or write data to a memory/port either 16 bits or 8 bit at a time
9. It can support up to 64K I/O ports.
10. It provides 14, 16-bit registers.
11. Executed instructions in as little as 400 ns (2.5 **millions of instructions per second)**
12. It is available in three versions:
    a. 8086 (5 MHz)
    b. 8086-2 (8 MHz)
    c. 8086-1 (10 MHz)



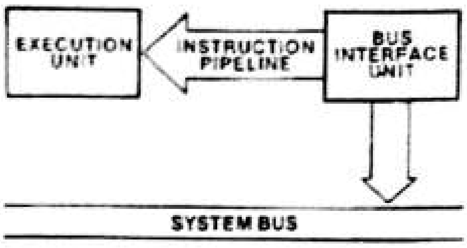
Pin diagram of 8086

◉Intel 8086 contains two independent functional units:

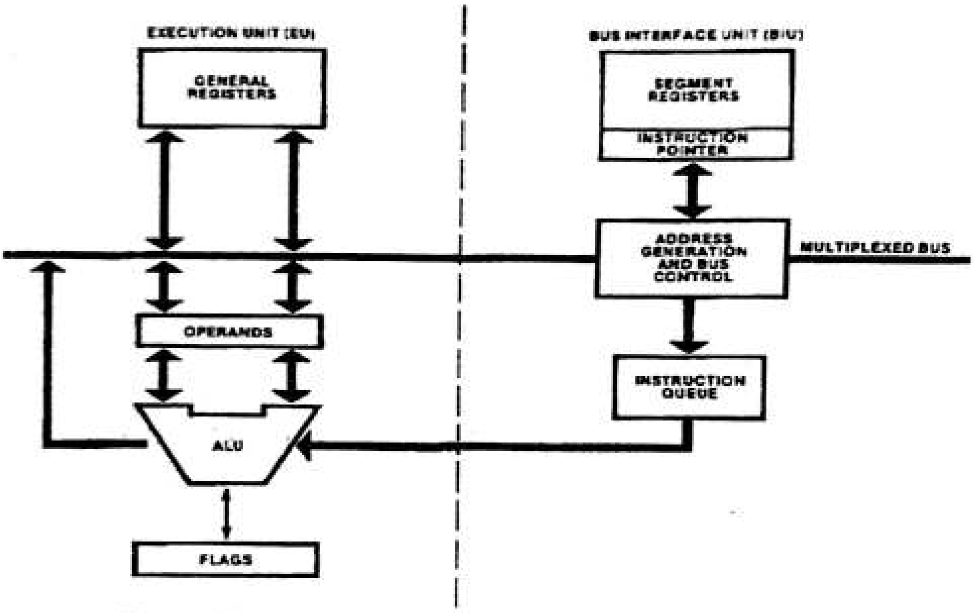
Intel 8086 contains two independent

◉Bus Interface Unit (BIU) ◉Execution Unit (EU)

Intel 8086 contains two independent, both units operate **asynchronously** to give the 8086 an overlapping instruction fetch and execution mechanism which is called as **Pipelining**. This results in efficient use of the system bus and system performance.



Pipelined architecture of the 8088/8086 microprocessor



Execution unit and Bus interface unit

3

The Bus Interface Unit (BIU) contains: - Bus Interface Logic, Segment registers, Memory addressing logic and a Six byte instruction object code queue (4-byte instruction object-code queue in case of 8088 microprocessor).

The **Execution Unit (EU) contains**: - Data and Address registers, the Arithmetic and Logic Unit and the Control Unit.

◉ Bus Interface Unit contains:

◉ Segment Registers

◉ 6-Byte Instruction Queue

◉ Instruction Pointer

◉ Address adder

◉ Execution Unit contains:

◉ General Purposes Registers

◉ Base Pointer

◉ ALU

◉ Instruction Decoder

◉ Stack Pointer

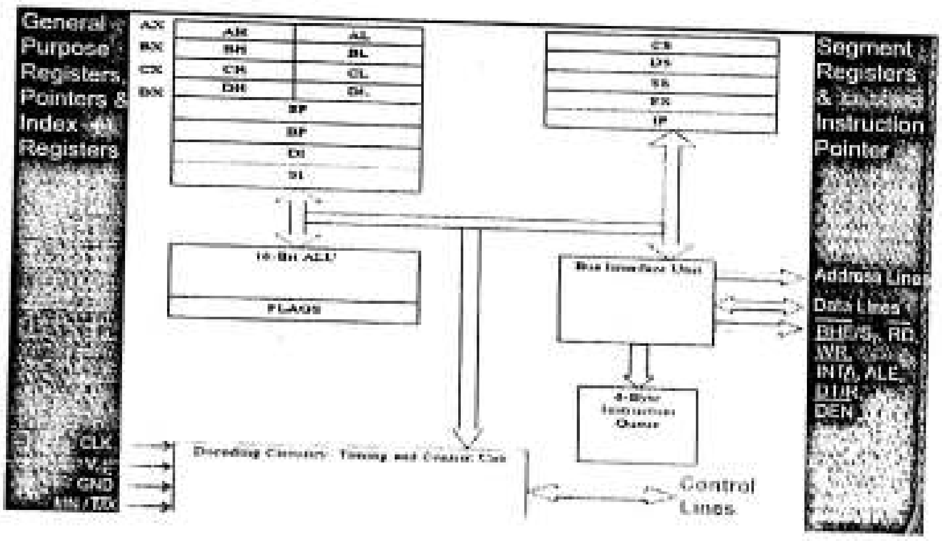
◉ Index Registers

◉ Flag Register

◉ Timing & Control Unit

Bus Interface Unit (BIU)

The BIU sends out addresses, fetches instructions from memory, reads data from memory and ports, and writes data to ports and memory. In other words the BIU handles all transfers of data and addresses on the buses for the execution unit.


General Purpose Registers, Pointers & Index Registers
AX
BX
CX
DX
AH
AL
BH
BL
CH
CL
DH
DL
SP
BP
DI
SI
CS
DS
SS
ES
IP
Segment Registers & Instruction Pointer
16-Bit ALU
FLAGS
Bus Interface Unit
Address Lines
Data Lines
BHE/S, RD, WR, INTA, ALE, DT/R, DEN
6-Byte Instruction Queue
CLK
Vcc
GND
MN / MX
Decoding Circuitry, Timing and Control Unit
Control Lines

4

The function of BIU is to:

1. Connects to 'outside' world

2. It handles transfer of data and addresses between the processor and memory / IO.

3. It reads data from memory and I/O devices.

4. It writes data to memory and I/O devices.

5. It fetches instruction codes.

5. It stores fetched instruction codes in a FIFO register called QUEUE.

7. Idle state is when BIU does not prefetch any instruction; Queue is full and EU is not requesting

8. When the Q is not full OR EU is not asking for R?W from memory , the BIU will prefetch the next instruction in FIFO manner.

9. Calculating the addresses of the memory operands.

The BIU also contains a dedicated adder which is used to generate the 20 bit **physical address** that is output on the address bus. This address is formed by adding an ***appended 16 bit segment address*** and a ***16 bit offset address***.

The BIU has

I-. An instruction queue II. An Instruction pointer III. Segment registers

I- Instruction Queue

1. To increase the execution speed, BIU fetches as many as six instruction bytes ahead to time from memory.
2. All six bytes are then held in first in first out 6 byte register called instruction queue.
3. Then all bytes have to be given to EU one by one.
4. This pre fetching operation of BIU may be in parallel with execution operation of EU, which improves the speed execution of the instruction.

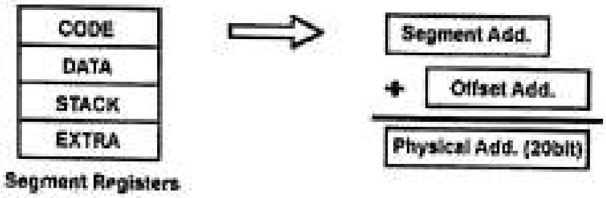
5

II- Instruction Pointer (IP)

- The Instruction Pointer (IP) in 8086 acts as a Program Counter (PC).
- It points to the address of the next instruction to be executed.
- Its content is automatically incremented when the execution of a program proceeds further.
- The actual address is obtained by combining its content with CS register value of next code is CS:IP
- This is done during the Fetch Cycle.

III- Segment Registers

The 8086 / 8088 microprocessor has 20-bit address lines. All the registers in 8086 / 8088 are 16-bits in length. Hence to obtain 20-bit addresses from the available 16-bit registers, all 8086 / 8088 memory addresses are computed by summing the contents of a segment register and an effective memory address. The process of adding, to obtain 20-bit address ***(Will be discussed later).***



- Additional registers called **segment registers** generate memory address when combined with other in the microprocessor.
- A segment register points to the starting address of a memory segment.

The **code segment register** points to the starting address of the code segment.

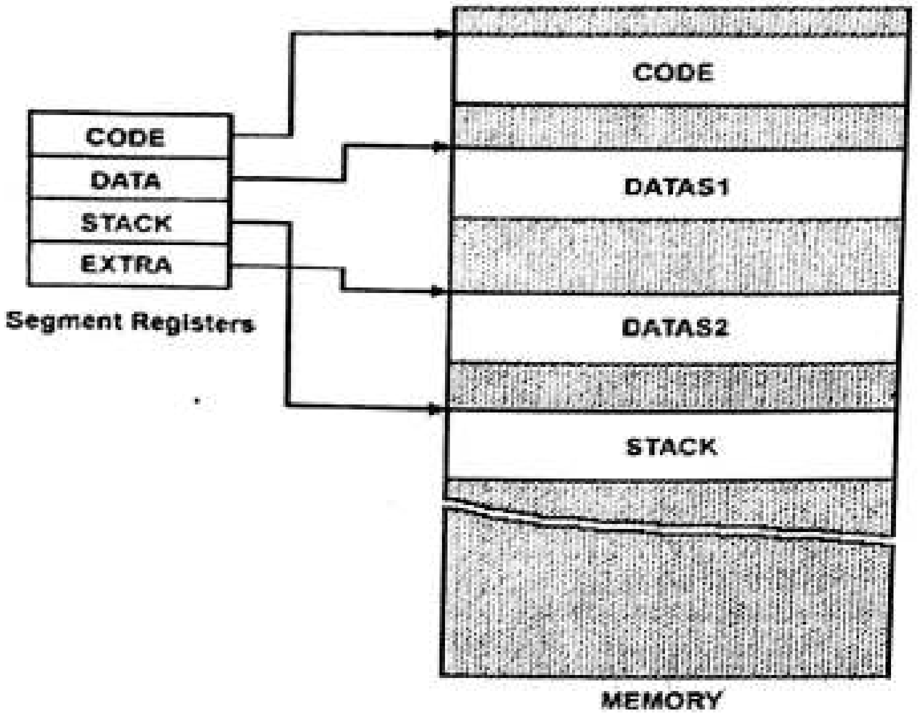
The **data segment register** points to the starting address of the data segment.

The **Stack Segment register** points to the starting address of the stack segment

The **Extra Segment register** points to the starting address of the data segment

6

- The maximum capacity of a segment may be up to 64 KB.



In 8086 microprocessor, memory is divided into 4 segments as follow:

1. **Code Segment (CS):** The CS register is used for addressing a memory location in the Code Segment of the memory, where the executable program is stored.

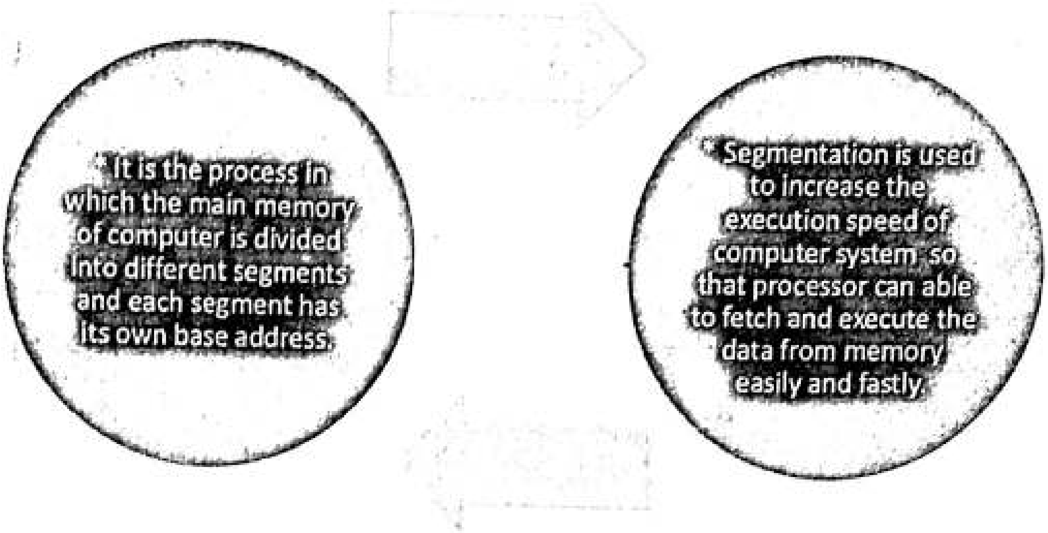
2. **Data Segment (DS):** The DS contains most data used by program. Data are accessed in the Data Segment by an offset address or the content of other register that holds the offset address.

3. **Stack Segment (SS):** SS defined the area of memory used for the stack.

4. **Extra Segment (ES):** ES is additional data segment that is used by some of the string to hold the destination data.

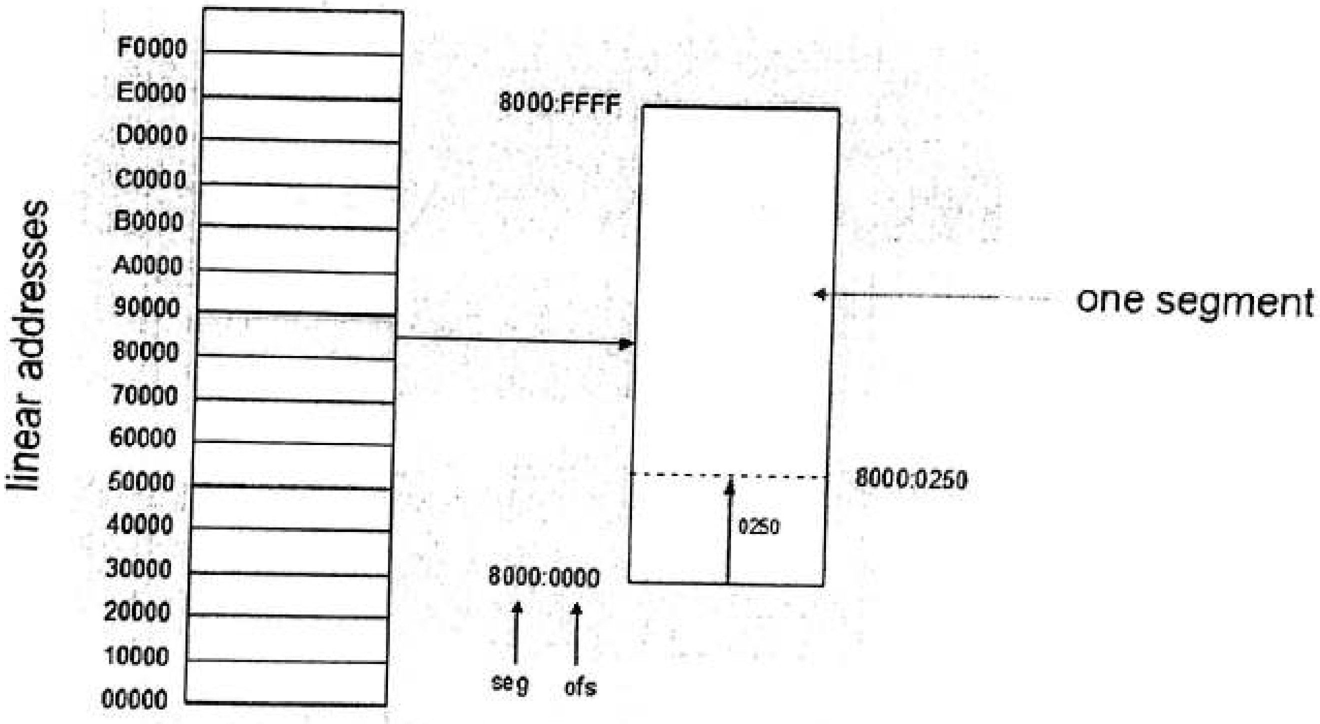
7

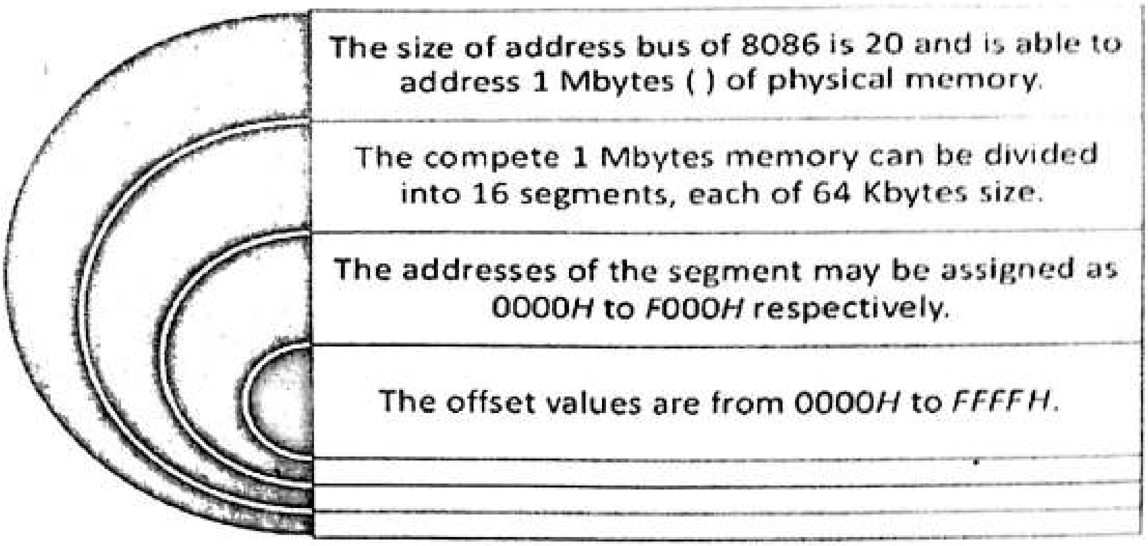
Segmentation



* It is the process in which the main memory of computer is divided into different segments and each segment has its own base address.

* Segmentation is used to increase the execution speed of computer system, so that processor can able to fetch and execute the data from memory easily and fastly.





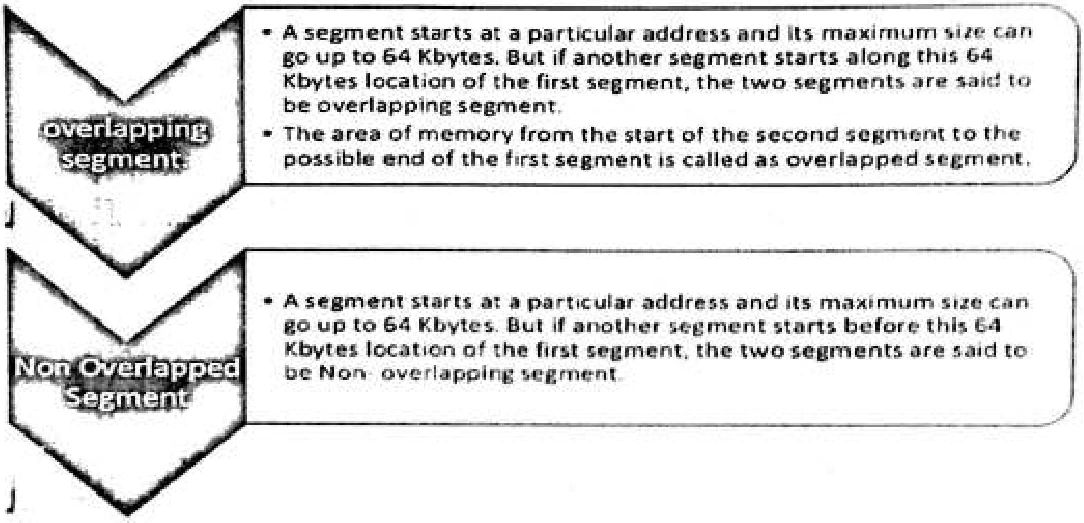
The size of address bus of 8086 is 20 and is able to address 1 Mbytes ( ) of physical memory.

The compete 1 Mbytes memory can be divided into 16 segments, each of 64 Kbytes size.

The addresses of the segment may be assigned as 0000*H* to F000*H* respectively.

The offset values are from 0000*H* to *FFFFH*.

Types of Segmentation

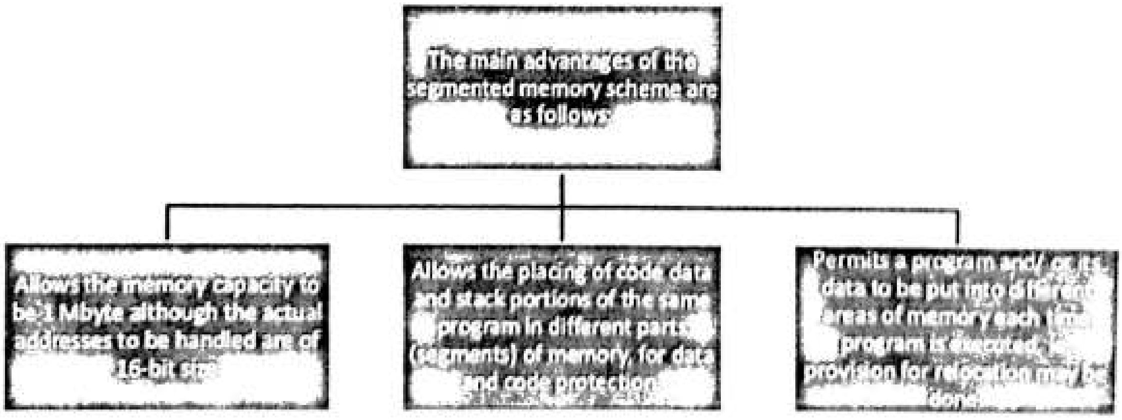


overlapping segment

- A segment starts at a particular address and its maximum size can go up to 64 Kbytes. But if another segment starts along this 64 Kbytes location of the first segment, the two segments are said to be overlapping segment.
- The area of memory from the start of the second segment to the possible end of the first segment is called as overlapped segment.

Non Overlapped Segment

- A segment starts at a particular address and its maximum size can go up to 64 Kbytes. But if another segment starts before this 64 Kbytes location of the first segment, the two segments are said to be Non- overlapping segment.

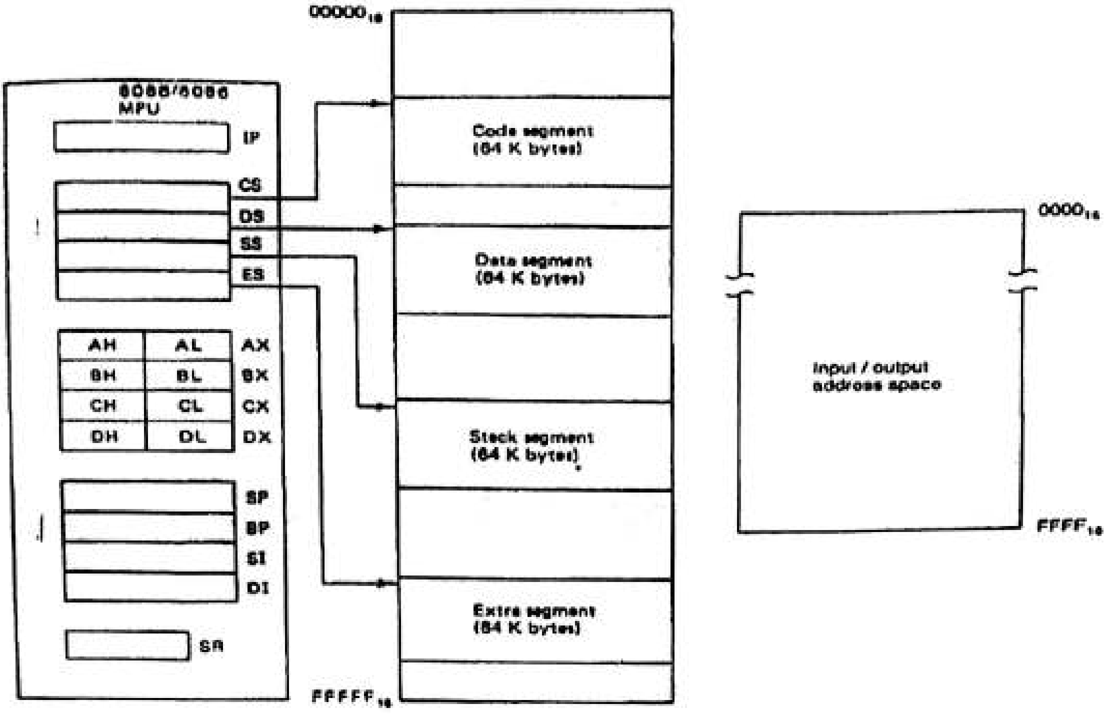


The main advantages of the segmented memory scheme are as follows:

Allows the memory capacity to be 1 Mbyte although the actual addresses to be handled are of 16-bit size.

Allows the placing of code data and stack portions of the same program in different parts (segments) of memory, for data and code protection.

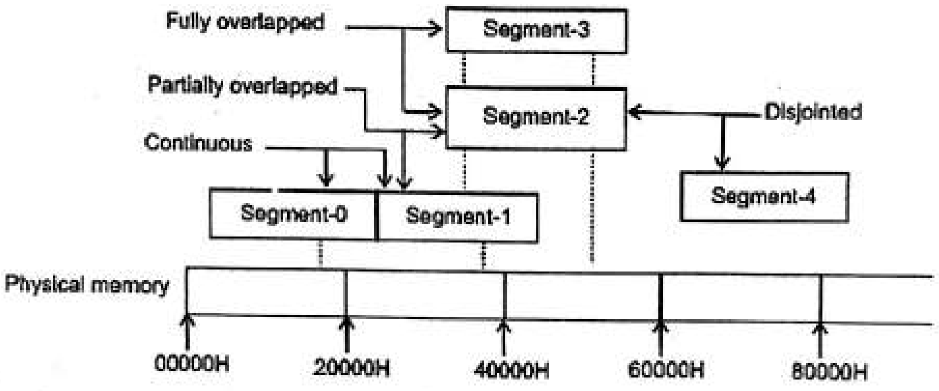
Permits a program and/ or its data to be put into different areas of memory each time program is executed, i.e. provision for relocation may be done.


00000₁₆
8088/8086 MPU
IP
CS
DS
SS
ES
AH
AL
AX
BH
BL
BX
CH
CL
CX
DH
DL
DX
SP
BP
SI
DI
SR
Code segment (64 K bytes)
Data segment (64 K bytes)
Stack segment (64 K bytes)
Extra segment (64 K bytes)
FFFFF₁₆
0000₁₆
Input / output address space
FFFF₁₆

The different memory segmentations done in case of 8086 are

- Continuous
- partially overlapped
- fully overlapped and
- disjointed

This is shown in Fig.



Depiction of different types of segments

In the figure,

Segments-0 and 1 → Continuous
Segments-1 and 2 → Partially overlapped
Segments-2 and 3 → Fully overlapped
and Segments-2 and 4 → Disjointed

Execution Unit (EU)

The functions of execution unit are:

1. It receives opcode of an instruction from the QUEUE.

2. It decodes it and then executes it.

3. It tells BIU where to fetch the instructions or data from.

4. It contains the control circuitry to perform various internal operations.

5. It has 16-bit ALU, which can perform arithmetic and logical operations on 8-bit as well as 16-bit data.

6. Update the status and control flag

7. Generate address (if necessary) – pass it to BIU and request BIU to fetch the data from the memory.

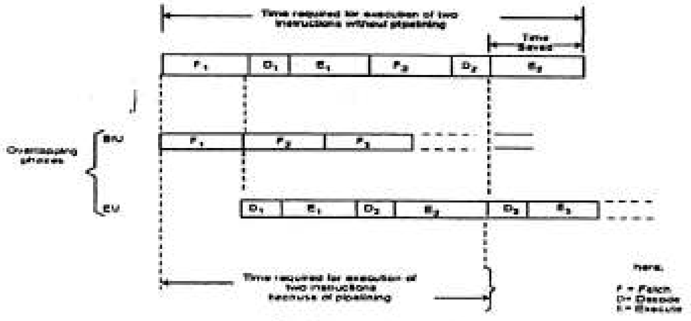
8. If Q is empty, EU waits for the next instruction byte to be fetched by BIU.

Pipelining

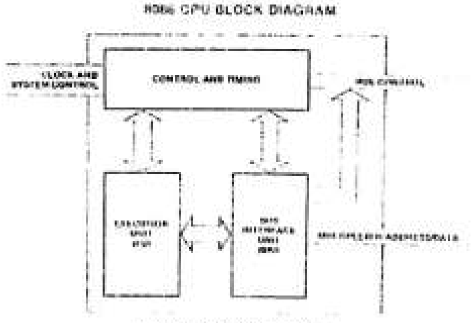
- While EU executes instructions, BIU fetches instructions from memory and stores them in the QUEUE.

- BIU and EU operate in parallel independent of each other.

- This type of overlapped operation of the functional units of a MP is called Pipelining.



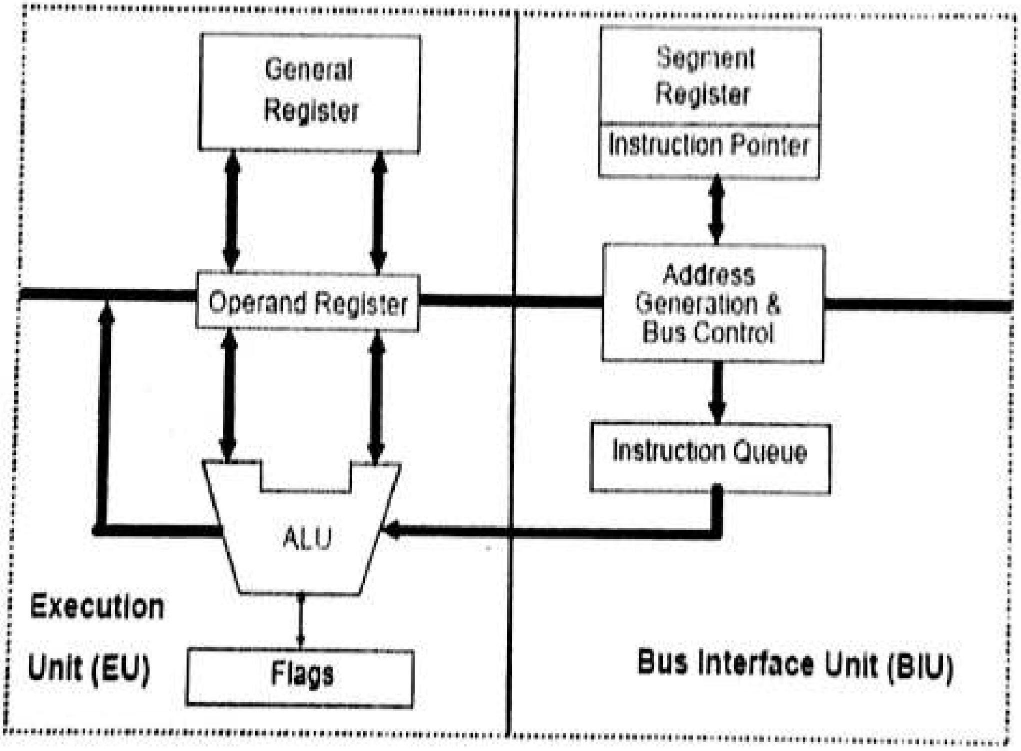
Pipelining procedure saves time

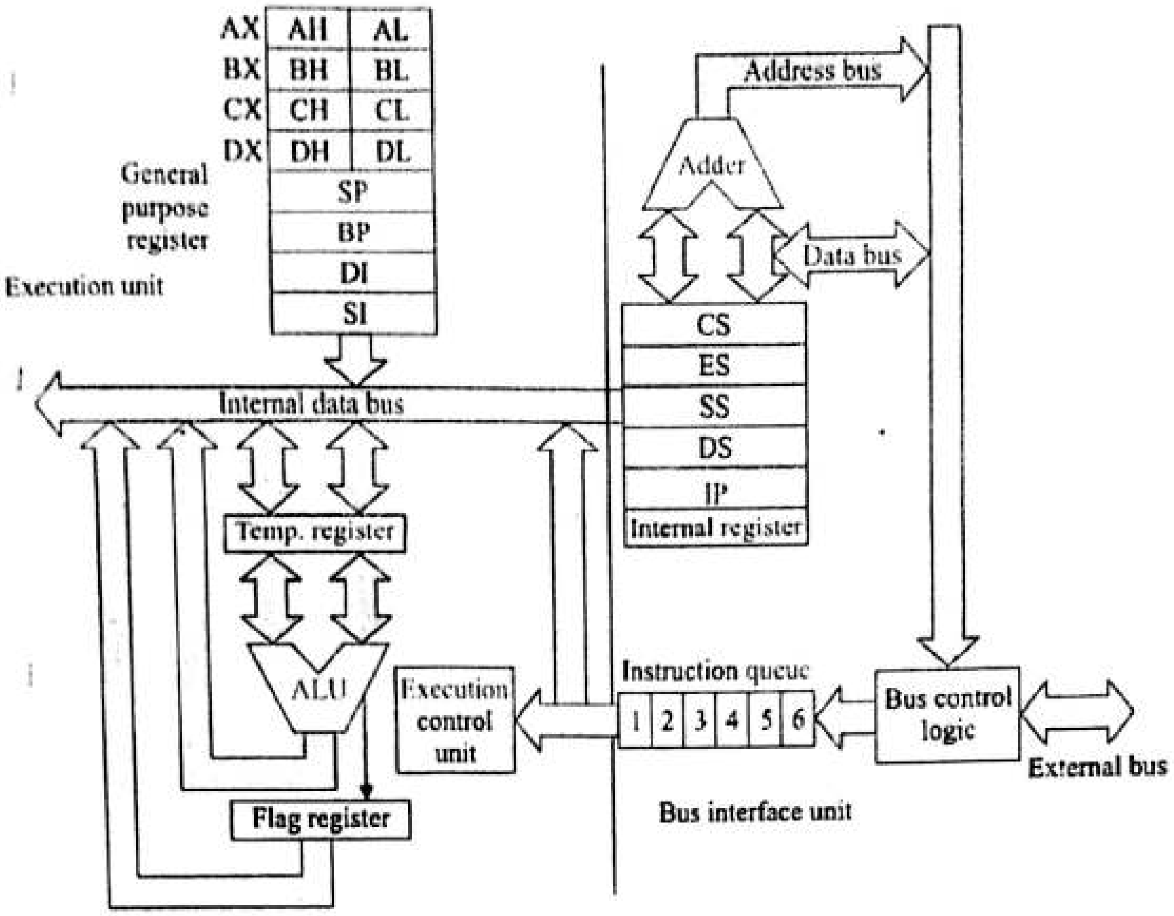


TWO INDEPENDENT UNITS, EU AND BIU
PARALLEL OPERATION
[illegible]

How do BIU and EU work?

- Fetch and execute cycles overlap
  - BIU outputs the contents of the IP onto the address bus
  - Register IP is incremented by one or more than one for the next instruction fetch
  - Once inside the BIU, the instruction is passed to the queue; this queue is a first-in-first-out register sometimes likened to a pipeline
  - Assuming that the queue is initially empty the EU immediately draws this instruction from the queue and begins execution
  - While the EU is executing this instruction, the BIU proceeds to fetch a new instruction.
    - BIU will fill the queue with several new instructions before the EU is ready to draw its next instruction
  - The cycle continues with the BIU filling the queue with instructions and the EU fetching and executing these instructions





Arithmetic Logic Unit (ALU)

ALU is 16-bits wide. It can do the following 16-bits arithmetic operations

(i) Addition (ii) Subtraction (iii) Multiplication (iv) Division

Arilhmetic operations may be performed on four types of numbers:-

Unsigned binary numbers. Signed binary numbers (Integers)

Unsigned packed decimal numbers Unsigned unpacked decimal numbers

The ALU can also perform logical operations such as

(i) NOT (ii) AND (iii) OR

(iv) EXCLUSIVE OR (v) TEST

XoR

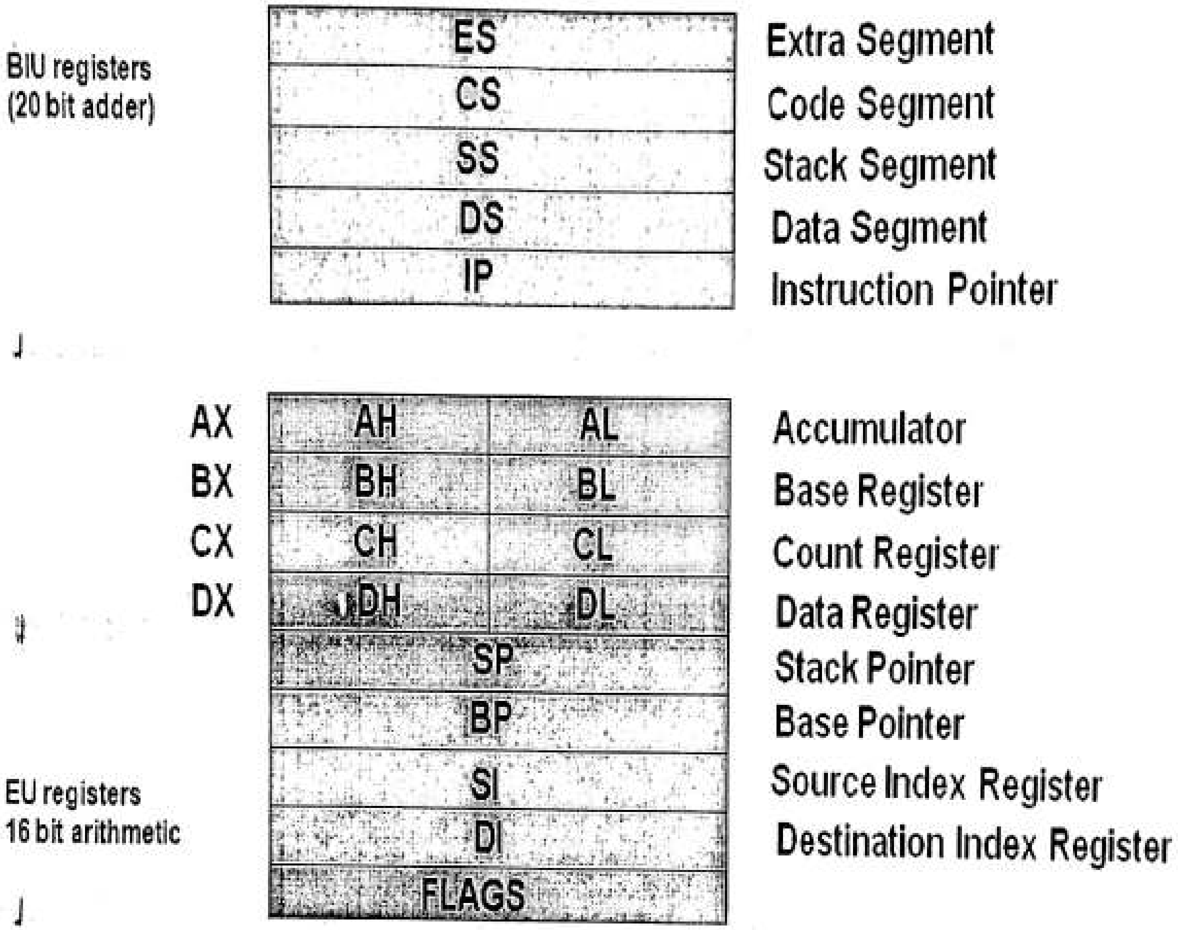
Registers of 8086

[14 16-bit registers]

In the CPU, registers are used store information temporarily. The information can be one or two bytes of data, or the address of data.

[AX , BX, CX, DX , CS , SS, DS, ES, SR, SP, BP, SI, DI, IP]

8086 Programmer's Model



BIU registers
(20 bit adder)

	ES		Extra Segment
	CS		Code Segment
	SS		Stack Segment
	DS		Data Segment
	IP		Instruction Pointer

AX	AH	AL	Accumulator
BX	BH	BL	Base Register
CX	CH	CL	Count Register
DX	DH	DL	Data Register
	SP		Stack Pointer
	BP		Base Pointer
	SI		Source Index Register
	DI		Destination Index Register
	FLAGS		

EU registers
16 bit arithmetic

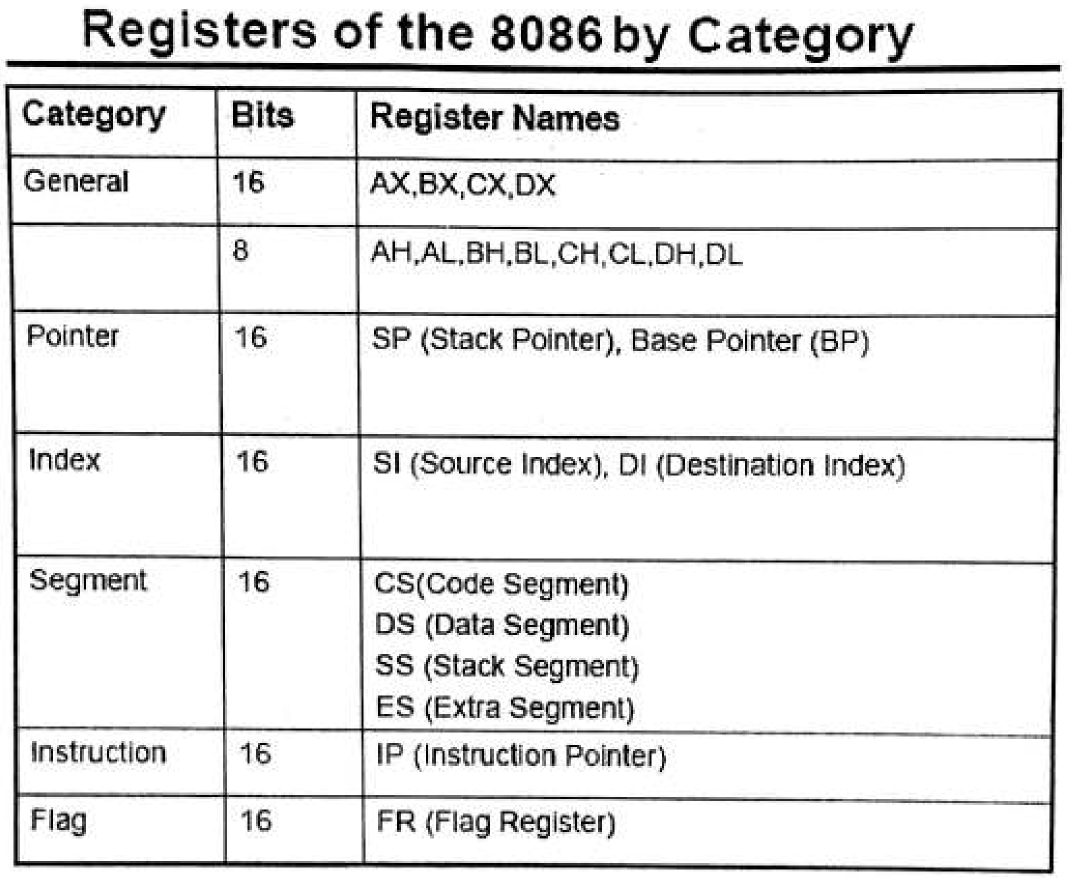
Two types

1- General Purpose Registers : AX , BX, CX, DX
Holding data, holding variables, store temporary for counting purpose, storing offset address.

2- Special Purpose Registers: CS , SS, DS, ES, SR, SP, BP, SI, DI, IP
Segment registers, pointers, index registers, program counter. Etc

Four categories

1. General data Registers
2. Segment Registers
3. Pointers, Index Registers
4. Flag registers



Registers of the 8086 by Category

Category	Bits	Register Names
General	16	AX,BX,CX,DX
	8	AH,AL,BH,BL,CH,CL,DH,DL
Pointer	16	SP (Stack Pointer), Base Pointer (BP)
Index	16	SI (Source Index), DI (Destination Index)
Segment	16	CS(Code Segment) DS (Data Segment) SS (Stack Segment) ES (Extra Segment)
Instruction	16	IP (Instruction Pointer)
Flag	16	FR (Flag Register)

General Purpose Registers of 8086 (Multipurpose registers)

- These registers can be used as 8-bit registers individually or can be used as 16-bit in pair to have AX, BX, CX, and DX. These registers hold various data sizes (byte, word).
- Each of these 16-bit registers are further subdivided into two 8-bit registers.

AX 16-bit register	
AH 8-bit reg.	AL 8-bit reg.

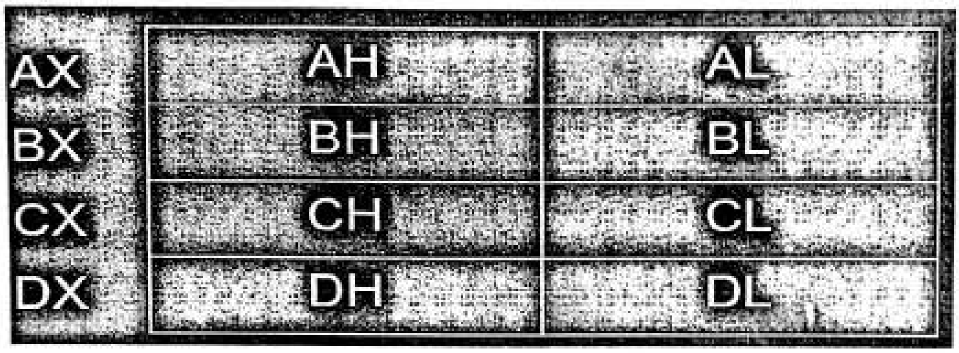
➢ The bits of the registers are numbered in descending order:

8-bit register:

D7	D6	D5	D4	D3	D2	D1	D0

16-bit register:

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0



AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

1. **AX Register:** AX register is also known as **accumulator register** that stores operands for arithmetic operation like divided, rotate. The accumulator is used for instruction such as multiplication, division, and some of the adjustment instruction

2. **BX Register:** This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment. The BX register some tomes holds the ***offset address*** of a location in the memory system.

3. **CX Register:** It is defined as a counter. Its holds the count for various instructions such as (repeated string instruction, shift, rotate and loop). The shift and rotate instructions use CL as the count, the repeated string instruction and loop instruction use CX.

4. **DX Register:** DX register is used to contain I/O port address for I/O instruction. DX holds a part of the result from a multiplication or part of the dividend before a division.

Special-Purpose Registers: Special-Purpose Registers include IP, SP, FLAGS and the segment register CS, DC ,ES and SS.

Base Pointer (BP)

Used to point data array in DATA Segment

Index Pointers (SI, DI)

Used primarily in instructions which deals with long strings of bytes that need to be moved from one block in a memory to another

1- Source Index Register (SI)

Used to point to a byte or word in the current data segment that needs to be fetched as a part of a block of data, this register is always used with the DATA segment, i.e. DS:SI = physical address (e.g. used with string copy instructions as the source address for data)

2- Destination Index Register (DI)

Similar to SI, but used as a destination address in the extra segment for a byte or a word that is part of a block of data being transferred,

ES:DI = physical address

Stack Pointer (SP):

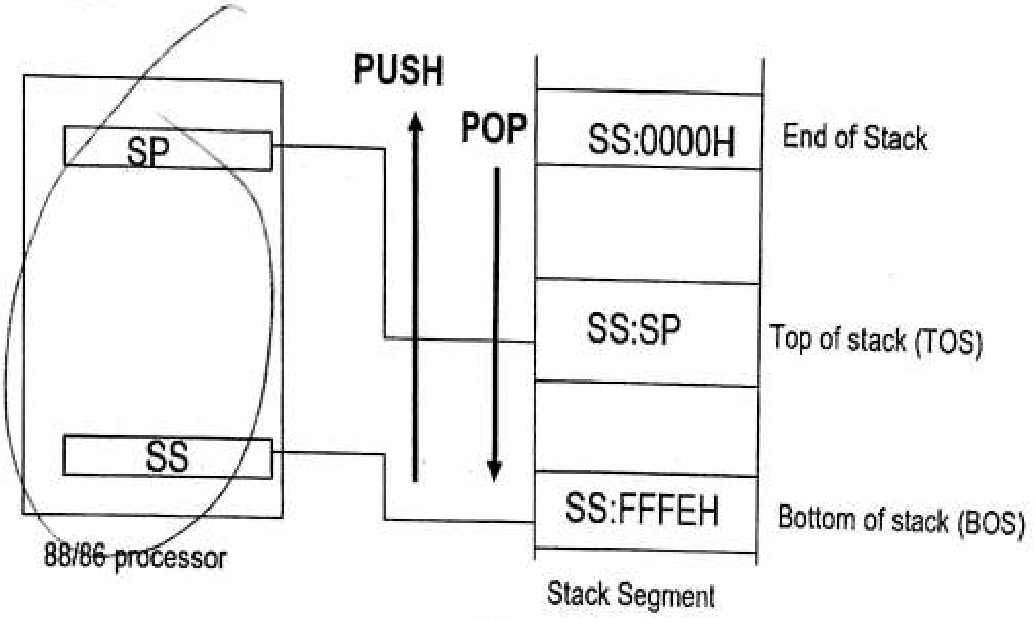
- Used in stack operation (?)
- Usage combined with SS
- The function of SP is stores the address of top element in the stack.

Stack memory

What is the stack?

The Stack is an area of memory for keeping temporary data.

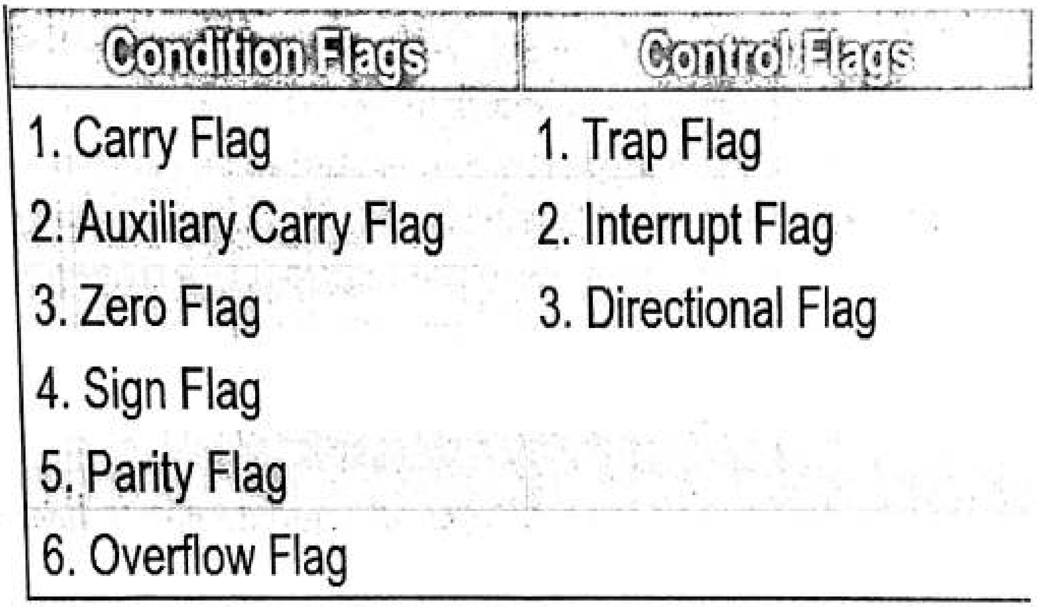
- The stack is 64 KB used for temporary storage of information.
- The Stack is a Last In First Out (LIFO) memory. Data is placed onto the Stack Organized from software point of view as 32 KWs.
- SP contains an offset value to the stack segment. Therefore, (SS:SP) is the physical address of 1 st storage location in the stack to which data were pushed. It is referred to as Top Of Stack (TOS)
- SP is initiated to FFFEH which is referred to as The Bottom of Stack (BOS)
- Processor pushes **one word** at a time.
- When a word is pushed, the SP is automatically decremented by 2 creating new location for two bytes and then the word is stored in this location. The reverse sequence occurs when data is POPPED from the Stack. The SP is incremented by 2.
- Stack grows from bottom of stack SS:FFFEH towards the end of stack SS:0000H



Flag Registers of 8086 [Status flags]

- 8086 has 16-bit status register.
- Status Flags determines the current state of the Accumulator.
- They are modified automatically by CPU after mathematical operations.
- This allows to determine the type of the result.
- Determine conditions to transfer control to other parts of the program
- It is also called Flag Register or Program Status Word (PSW).
- There are nine status flags and seven bit positions remain unused.
- 8086 has 9 flags and they are divided into two categories:
- Condition Flags Control Flags

				OF	DF	IF	TF	SF	ZF		AF		PF		CF
15															0



Condition Flags	Control Flags
1. Carry Flag	1. Trap Flag
2. Auxiliary Carry Flag	2. Interrupt Flag
3. Zero Flag	3. Directional Flag
4. Sign Flag	
5. Parity Flag	
6. Overflow Flag	

I- Conditional Flags

Conditional flags represent result of last arithmetic or logical instruction executed. Conditional flags are as follows:

1. **Carry Flag (CF):** This flag will be set to one if the addition of two 16-bit binary numbers produces a carry out of the most significant bit position or if there is a borrow to the MSB after subtraction. This flag is also affected when other arithmetic and logical instruction are executed. This flag indicates an overflow condition for unsigned integer arithmetic.

Auxiliary Flag (AF): If an operation performed in ALU generates a carry/barrow from lower nibble (i.e. $D_0 - D_3$) to upper nibble (i.e. $D_4 - D_7$), the AF flag is set i.e. carry given by D_3 bit to D_4 is AF flag. This is not a general-purpose flag, it is used internally by the processor to perform Binary to BCD conversion.

Parity Flag (PF): This flag is used to indicate the parity of result. If lower order 8-bits of the result contains even number of 1's, the Parity Flag is set and for odd number of 1's, the Parity Flag is reset. This flag can be used to check for data transmission error.

Zero Flag (ZF): It is set; if the result of arithmetic or logical operation is zero else it is reset.

Sign Flag (SF): This flag is set, when an MSB bit of the result is high after an arithmetic operation. When this flag is set the data in assumed to be negative and when this flag iszero it is assumed to be positive.

Overflow Flag (OF): Overflow means that the size of the result exceeded the storage capacity of the destination, and a significant digit has been lost. It occurs when signed numbers are added or subtracted. An OF indicates that the result has exceeded the capacity of machine.

II- Control Flags

Control flags are used to control certain operations of the processor. The application of these flags are different from that of six conditional flags. The conditional flags are set or reset by the EU on the basis of the result of some arithmetic or logic operations. The control flags are deliberately set or reset with specific instructions included in the program. Control flags are as follows:

1. **Trap Flag (TP):**
   - a. It is used for single step control.
   - b. It allows user to execute one instruction of a program at a time for debugging.
   - c. When trap flag is set, program can be run in single step mode.
2. **Interrupt Flag (IF):**
   - a. It is an interrupt enable/disable flag.
   - b. If it is set, the maskable interrupt of 8086 is enabled and if it is reset, the interrupt is disabled.
   - c. It can be set by executing instruction **STI** and can be cleared by executing **CLI** instruction.
3. **Direction Flag (DF):**
   - a. It is used in string operation.
   - b. If it is set, string bytes are accessed from higher memory address to lower memory address.
   - c. When it is reset, the string bytes are accessed from lower memory address to higher memory address.
   - d. It can be set by executing instruction **STD** and can be cleared by executing **CLD** instruction

❑ Direction Flag (DF) is used to control the way SI and DI are adjusted during the execution of a string instruction

— DF=0, SI and DI will auto-increment during the execution; otherwise, SI and DI

21

Example1

If the following addition is carried out,

```
  0010 0011 0100 0101
+ 0011 0010 0001 1001
---------------------
  0101 0101 0101 1110
```

the FLAGS register's condition flags are set as follows:

S (sign) =0	Z (zero) =0
P (parity) =0	C (carry) =0
A (aux carry) =0	O (overflow) =0

Example2

If jnstead the following were performed,

```
  0101 0100 0011 1001
+ 0100 0101 0110 1010
---------------------
  1001 1001 1010 0011
```

the FLAGS register's condition flags world be set as follows:

S (sign) =1.	Z (zero) =0
P (parity) =1	C (carry) =0
A (aux carry) =1,	O (overflow) =1

Ex: Show how the flag register is affected by the addition of 38H and 2FH.

```
Solution:      MOV  BH,38H               BH=38H

               ADD  BH,2FH               BH = BH + 2F =  38 + 2F= 67H

                     38        0011 1000
                  +  2F        0010 1111
               ------------------------
                     67        0110 0111
```

CF = 0 since there is no carry beyond d7

PF = 0 since there is odd number of 1's in the result

AF = 1 since there is a carry from d3 to d4

ZF = 0 since the result is not zero

SF = 0 since d7 of the result is zero

Ex: Show how the flag register is affected by the following addition

```
Solution:        MOV AX,34F5H          ;AX =34F5H

                 ADD AX,95EBH          ;AX = CAE0H

                  34F5        0011 0100 1111 0101

               +  95EB        1001 0101 1110 1011

                  CAE0        1100 1010 1110 0000
```

CF = 0 since there is no carry beyond d15

PF = 0 since there is odd number of 1s in the lower byte

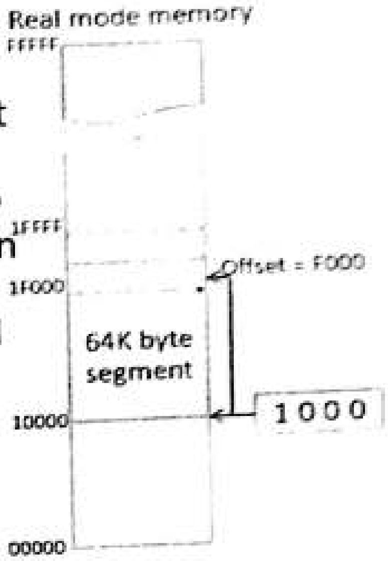
AF = 1 since there is a carry from d3 to d4

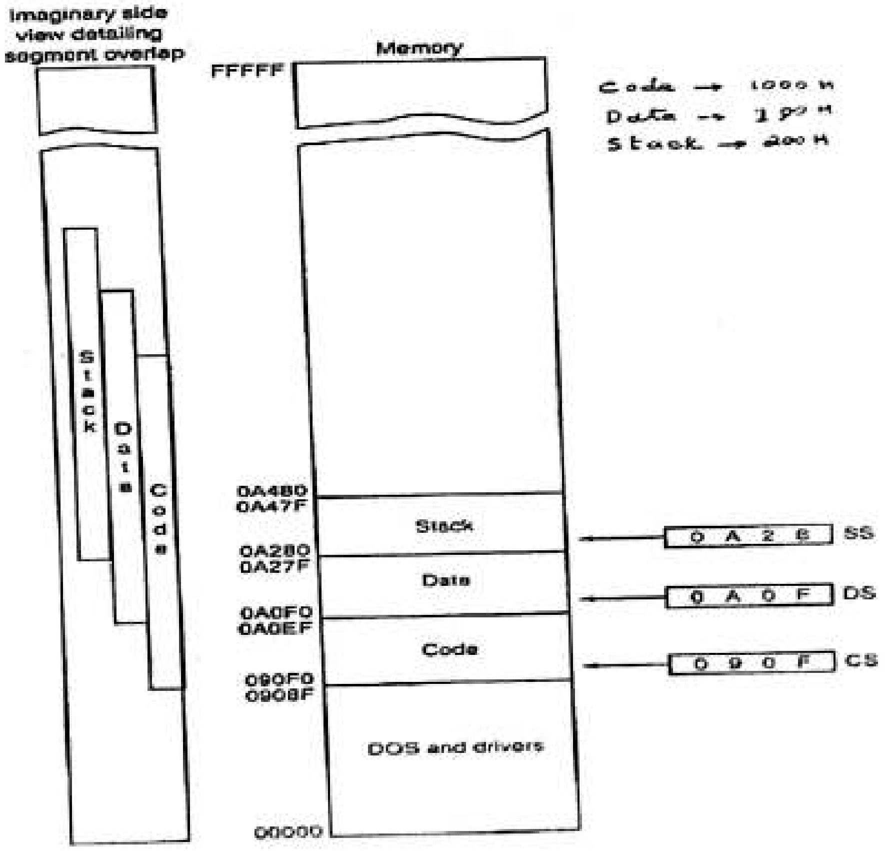
ZF ≠ 0 since the result is not zero

SF = 1 since d15 of the result is 1

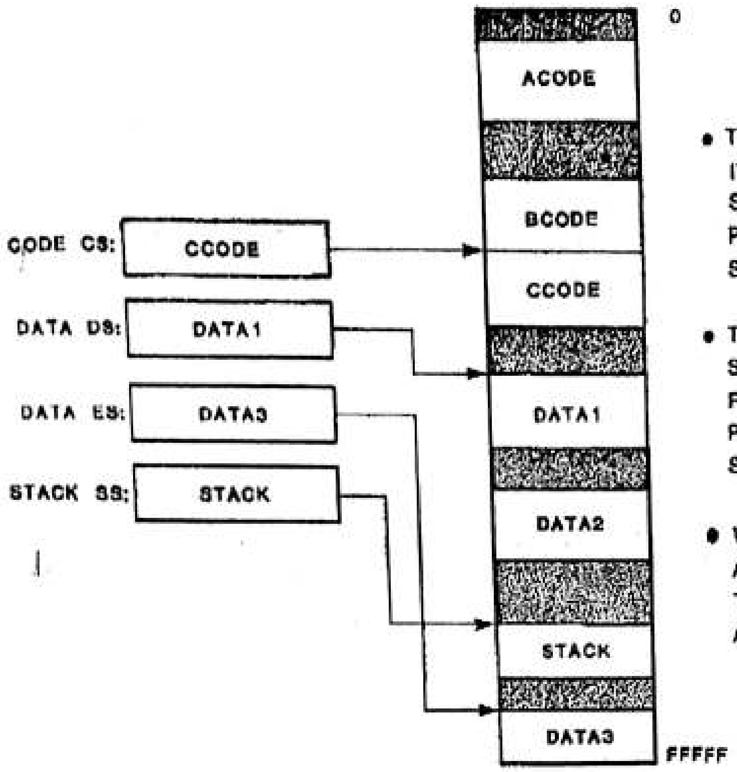
Real Mode Memory Addressing

- Location = Segment + Offset
  - Segment address located in a segment register; always appended with 0H
  - Segments always have length of 64 Kb
  - Offset or displacement selects location within 64 Kb of segment
  - e.g. 1000:2000 gives location 12000H
- Default Segment and Address Registers
  - e.g. code segment and instruction pointer CS:IP and stack segment and stack pointer SS:SP

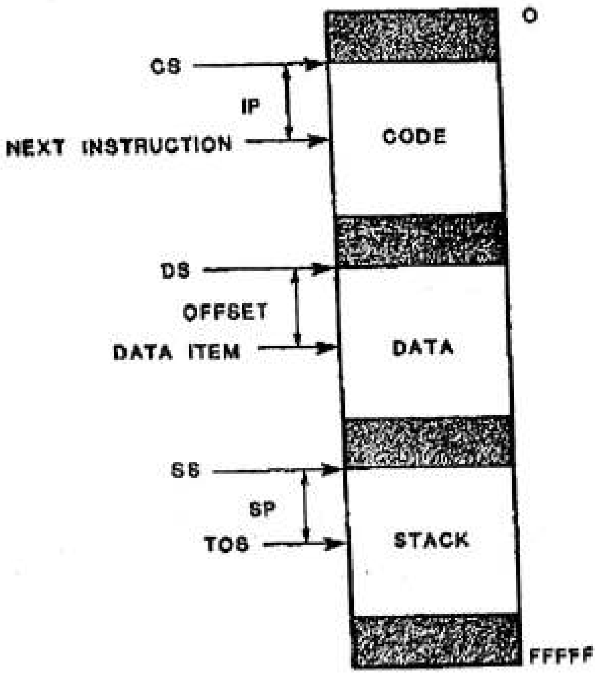




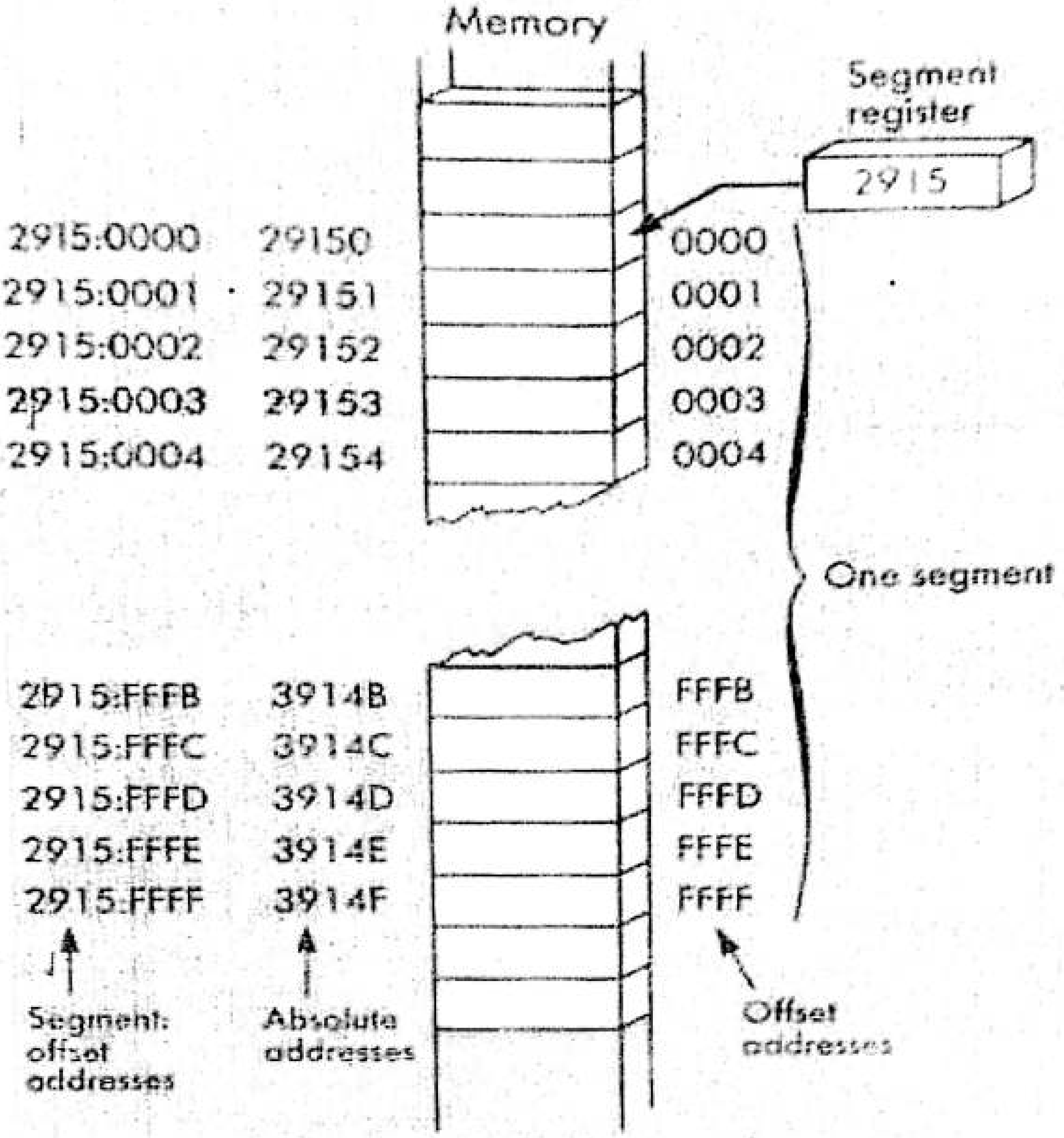
Code → 1000 H
Data → 190 H
Stack → 200 H



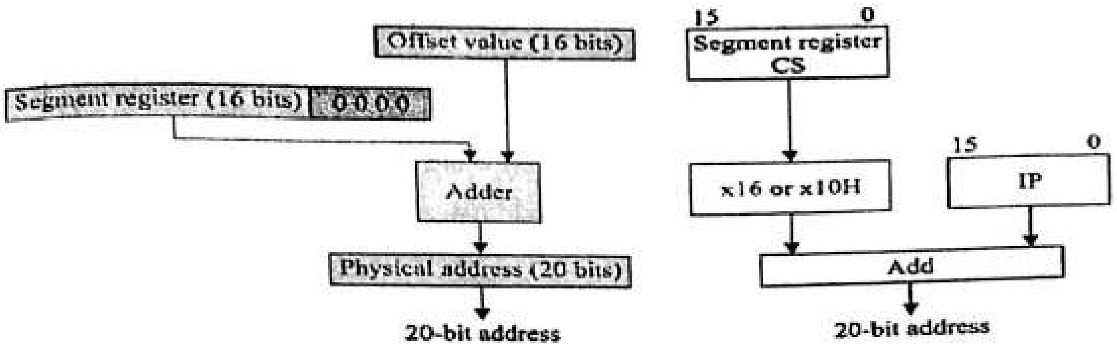
- THE 8086 CAN ACCESS ANY ITEM THAT RESIDES IN A SEGMENT CURRENTLY POINTED TO BY ONE OF THE SEGMENT REGISTERS.
- TO ACCESS ITEMS IN OTHER SEGMENTS, THE SEGMENT REGISTER IS CHANGED TO POINT TO THE OTHER SEGMENT
- WHAT IS THE MAXIMUM AMOUNT OF MEMORY THAT THE 8086 CAN ACCESS AT ANY GIVEN INSTANT?

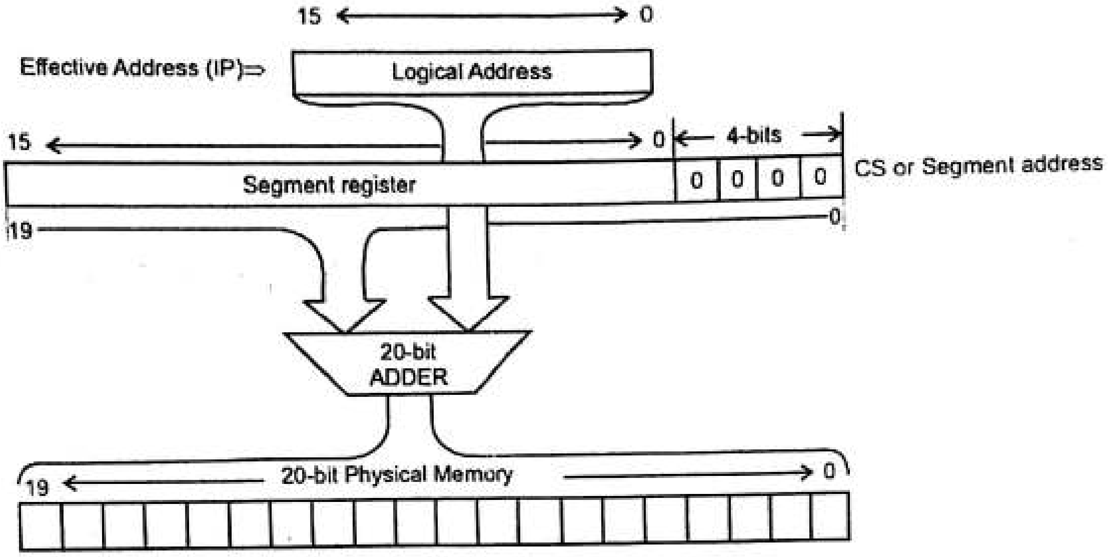


25


Memory
Segment register
2915
2915:0000
29150
0000
2915:0001
29151
0001
2915:0002
29152
0002
2915:0003
29153
0003
2915:0004
29154
0004
One segment
2915:FFFB
3914B
FFFB
2915:FFFC
3914C
FFFC
2915:FFFD
3914D
FFFD
2915:FFFE
3914E
FFFE
2915:FFFF
3914F
FFFF
Segment: offset addresses
Absolute addresses
Offset addresses

GENEMTING A MEMORY ADDREES





Physical address generation

Physical Address = Segment Register content 16 D + Offset.

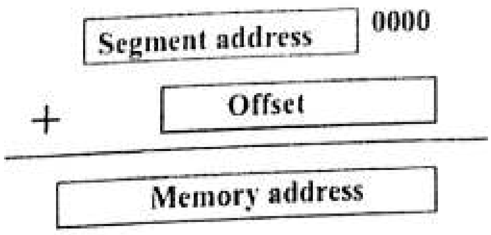
Offset Registers for various Segments

Segment register	CS	DS	ES	SS
Offset register(s)	IP	SI, DI, BX	SI, DI, BX	SP, BP

Scanned by CamScanner

Memory Address Calculation

- ❑ Segment addresses must be stored in segment registers
- ❑ Offset is derived from the combination of pointer registers, the Instruction Pointer (IP), and immediate values



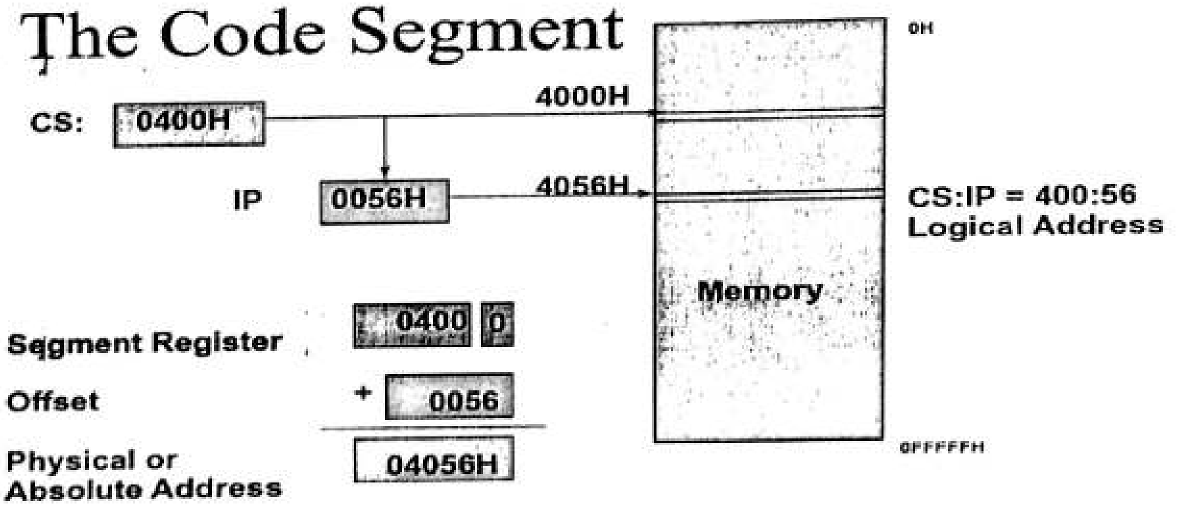
- ❑ Examples

CS	3	4	8	A	0
IP +		4	2	1	4
Instruction address	3	8	A	B	4

SS	5	0	0	0	0
SP +		F	F	E	0
Stack address	5	F	F	E	0

DS	1	2	3	4	0
DI +		0	0	2	2
Data address	1	2	3	6	2

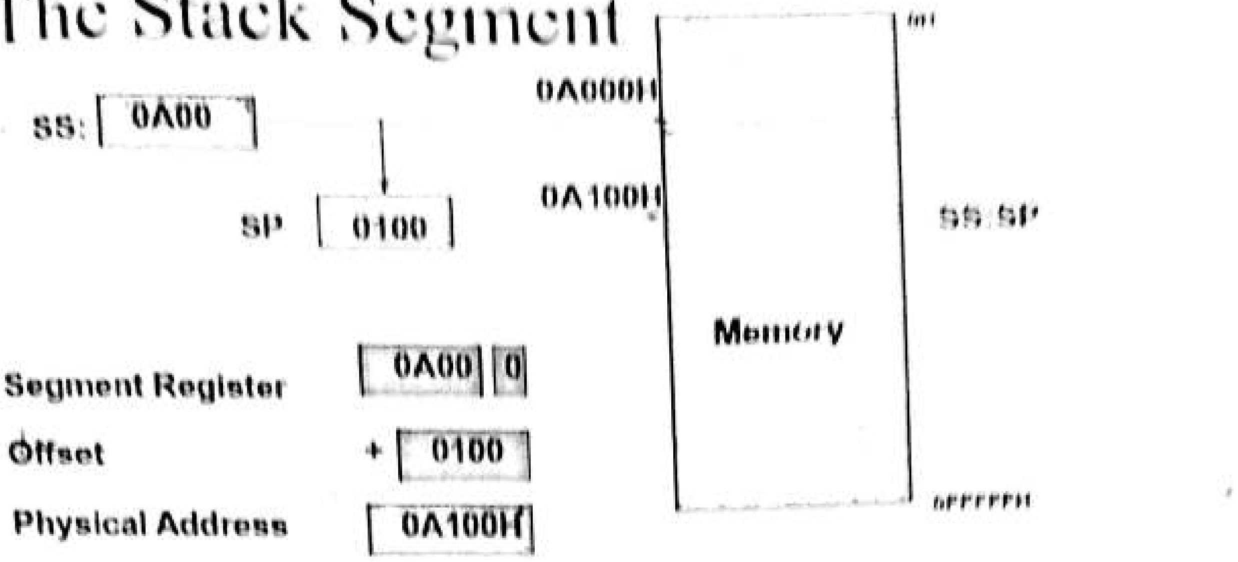
The Code Segment



The offset is the distance in bytes from the start of the segment.
The offset is given by the IP for the Code Segment.
Instructions are always fetched with using the CS register.

The physical address is also called the absolute address.

The Stack Segment


SS: 0A00

SP 0100

0A000H

0A100H

SS:SP

Memory

0H

0FFFFFH

Segment Register 0A00 0

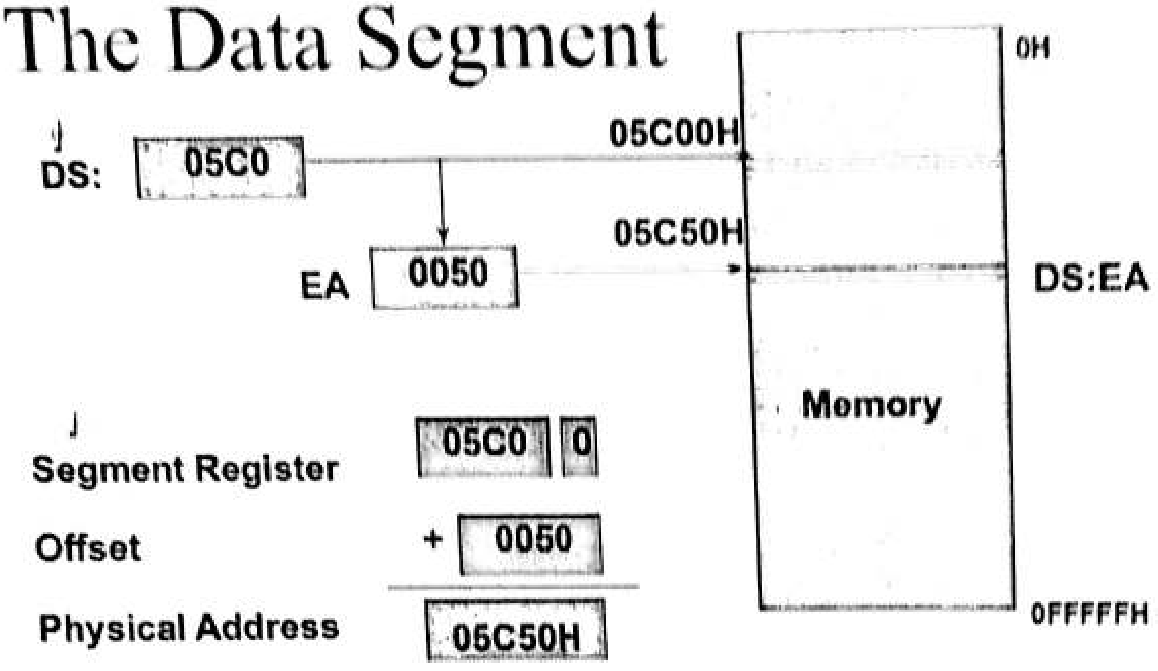
Offset + 0100

Physical Address 0A100H

The offset is given by the SP register.
The stack is always referenced with respect to the stack segment register.
The stack grows toward decreasing memory locations.
The SP points to the last or top item on the stack.

PUSH - pre-decrement the SP
POP - post-increment the SP

The Data Segment


DS: 05C0

EA 0050

05C00H

05C50H

DS:EA

Memory

0H

0FFFFFH

Segment Register 05C0 0

Offset + 0050

Physical Address 05C50H

Data is usually fetched with respect to the DS register.
The effective address (EA) is the offset.
The EA depends on the addressing mode.

Show how the code resides physically in the memory

CS:IP	Machine Language	Mnemonics
1132:0100	B057	MOV AL,57h
1132:0102	B686	MOV DH,86h
1132:0104	B272	MOV DL,72h
1132:0106	89D1	MOV CX,DX
1132:0108	88C7	MOV BH,AL
1132:010A	B39F	MOV BL,9F
1132:010C	B420	MOV AH,20h
1132:010E	01D0	ADD AX,DX
1132:0110	01D9	ADD CX,BX
1132:0112	05351F	ADD AX, 1F35h

Logical and Physical Address

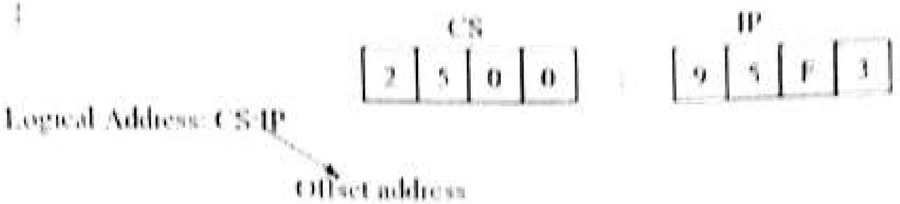
Physical Address is the 20-bit address that actually put on the address bus. Has a range of **00000H – FFFFFH**

Offset Address is a location within 64K byte segment range. Has a range of 0000H – FFFFH

Logical Address consists of segment address and offset address.

Addressing in Code segment

To execute a program, the 8086 fetches the instructions from the code segment. The logical address of an instruction consists **CS** (Code Segment) and **IP**(instruction pointer).



– ***Physical Address*** is generated by shifting the CS one hex digit to the left and adding IP.

Example: **CS:IP => 2500:95F3H**

1. **Start with CS** 2500
2. **Shift left CS** 25000
3. **Add IP** 2E5F3 (25000+95F3)

The microprocessor will retrieve the instruction in turn memory locations starting from 2E5F3.

Ex: If CS=24F6H and IP=634AH, determine:

a) The logical address
b) The offset address
c) The physical address
d) The lower range of the code segment
e) The upper range of the code segment

a) The logical address is: 24F6:634A
b) The offset address is:634A
c) The Physical address is:24F60+634A= 2B2AA
d) The lower range of the code segment: 24F6:0000 => 24F60+0000 =24F60
e) The upper range of the code segment: 24F6:FFFF => 24F60+FFFF=34F5F

Addressing in Data segment

The area of memory allocated strictly for data is called data segment. Just as the code segment is associated with CS and IP as segment register and offset. The data segment uses DS and an offset value. In 8086 BX, SI and DI are used to hold the offset address.

Ex: If DS=7FA2H and the offset is 438EH, determine:

a) The physical address

b) The lower range of the data segment

c) The upper range of the data segment

d) Show the logical address

a) The Physical address is; 7FA20+438E= 83DAE

b) The lower range: 7FA20(7FA20+0000)

c) The upper range: 8FA1F(7FA20+FFFF)

d) The logical address is; 7FA2:438E

Using the data segment

Assume that a program is needed to add 5 bytes of data (25H, 12H, 15H,1FH and 2BH)

```
One way:          MOV  AL,00H          ;initialize AL
                  ADD  AL,25H
                  ADD AL,12H
                  ADD AL,15H
                  ADD AL,1FH
                  ADD AL,2BH            AL=25+12+15+1F+2B
```

Other way: Assume that the offset for data segment begins at 0200H

```
DS:0200 = 25
DS:0201 = 12
DS:0202 = 15
DS:0203 = 1F
DS:0204 = 2B
```

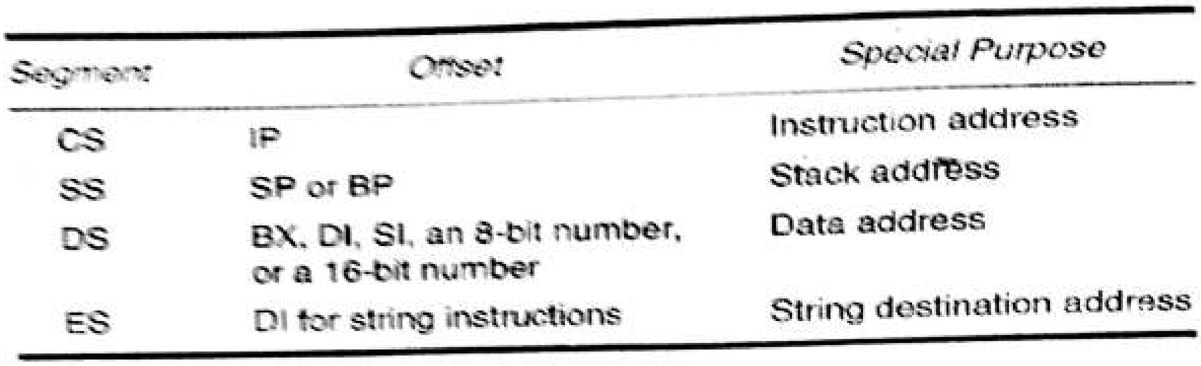
SUMMARY OF REAL MODE MEMORY ADDRESSING

- allows addressing of only 1M byte of memory space . The first 1M byte of memory is called the **real memory, conventional memory**.

- All real mode memory addresses must consist of a **segment address** plus an **offset address**.

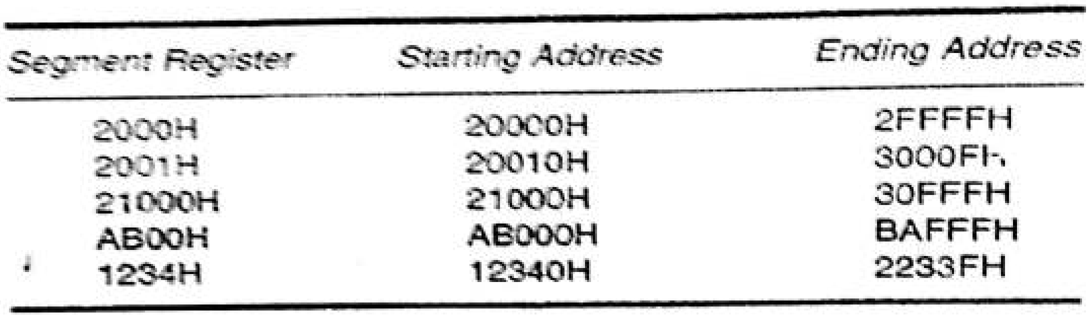
* **Segment address** :defines the beginning address of any 64K-byte memory segment.

* **Offset address**: selects any location within the 64K byte memory segment.



Segment	Offset	Special Purpose
CS	IP	Instruction address
SS	SP or BP	Stack address
DS	BX, DI, SI, an 8-bit number, or a 16-bit number	Data address
ES	DI for string instructions	String destination address

- Once the beginning address is known, the ending address is found by adding FFFFH. Because a real mode segment of memory is64K in length.



Segment Register	Starting Address	Ending Address
2000H	20000H	2FFFFH
2001H	20010H	3000FH
21000H	21000H	30FFFH
AB00H	AB000H	BAFFFH
1234H	12340H	2233FH

- The offset address is always added to the segment starting address to locate the data.

- Segment and offset address is sometimes written as **1000:2000** (**Logical address**) .

  * a segment address of 1000H; an offset of 2000H

  * **Physical address = segment address*(10)h + Offset address.**